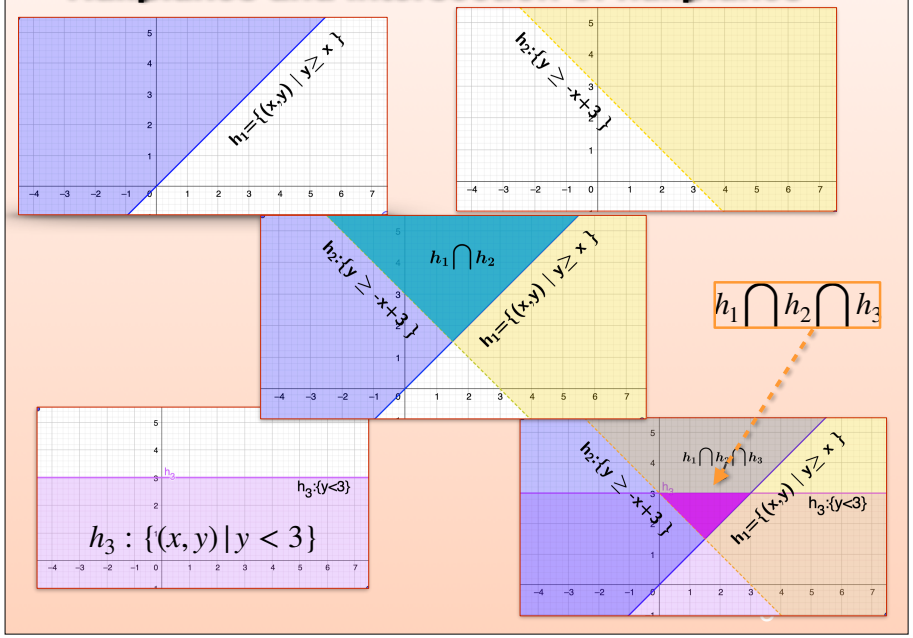
Linear Programming, healthy diets and ILP



Example of an LP: The Diet problem

- In the diet problem, we will have to compute two values x and y.
- x indicates how many **bananas** we plan to consume daily
- y indicates how many **oranges** we plan to consume daily
- The goal is to find a healthy diet that is as cheap as possible.

Halfplanes and intersection of halfplanes



Dot product notation (review from Linear Algebra)

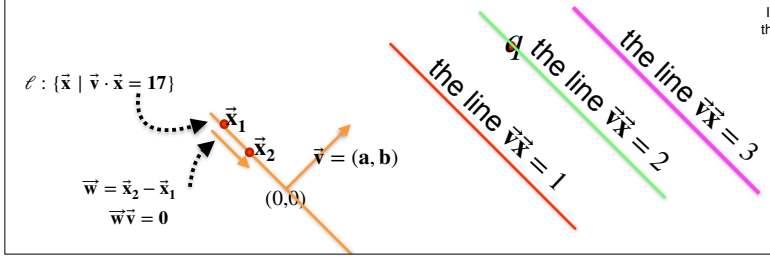
- In out context, a **vector** $\vec{\mathbf{v}}$ in the d-dimension space, is an ordered list of *d* numbers $\vec{\mathbf{v}} = (v_1 \ldots v_d)$.
- For two vectors, $\vec{\mathbf{v}} = (v_1, v_2, v_3, \ldots v_d)$ and $\vec{\mathbf{u}} = (u_1, u_2 . u_3, \ldots, u_d)$, we define the **dot product** $\vec{\mathbf{u}} \cdot \vec{\mathbf{v}}$ as follows:

  $\vec{\mathbf{u}} \cdot \vec{\mathbf{v}} = u_1 v_1 + u_2 v_2 + \ldots + u_d u_d = \sum_{i=1}^{d} u_i v_i$
- Note: $\vec{\mathbf{u}} \cdot \vec{\mathbf{u}} = \vec{\mathbf{v}} \cdot \vec{\mathbf{u}}$, and $\vec{\mathbf{u}} \cdot (\vec{\mathbf{v}} + \vec{\mathbf{w}}) = \vec{\mathbf{u}} \cdot \vec{\mathbf{v}} + \vec{\mathbf{u}} \cdot \vec{\mathbf{w}}$.
- The length of the vector $\vec{\mathbf{v}}$, denoted $|\vec{\mathbf{v}}|$ is $\sqrt{\vec{\mathbf{v}} \cdot \vec{\mathbf{v}}}$ (Pythagoras).
- $\vec{\mathbf{u}} \cdot \vec{\mathbf{v}} = \vec{\mathbf{v}} \cdot \vec{\mathbf{u}}$, and $\vec{\mathbf{u}} \cdot (\vec{\mathbf{v}} + \vec{\mathbf{w}}) = \vec{\mathbf{u}} \cdot \vec{\mathbf{v}} + \vec{\mathbf{u}} \cdot \vec{\mathbf{w}}$.

- Dot product strongly correlated to the angle between the vectors. If $\vec{\mathbf{u}} \cdot \vec{\mathbf{v}} = \mathbf{0}$, then they are orthogonal to each other.

- We distinguish between a vector and a **scalar.** A scalar is a single number, while a vector is a list of numberS.
- Let $\vec{\mathbf{v}} = (a, b)$. We can (sometimes) think about it as an arrow from the point $(0,0)$ to the point (a, b).
- Fix $\vec{\mathbf{v}} = (a, b)$. Think about all the points $\vec{\mathbf{x}} = (x, y)$ for which $\vec{\mathbf{v}} \cdot \vec{\mathbf{x}} = a \cdot x + b \cdot y = 0$. These points form a line ℓ. We can write $\ell := \{\vec{\mathbf{x}} \mid \vec{\mathbf{v}} \cdot \vec{\mathbf{x}} = \mathbf{0}\}$, or sometimes abbreviated as $\ell : \vec{\mathbf{x}} \cdot \vec{\mathbf{v}} = 0$
- The line ℓ is orthogonal to $\vec{\mathbf{v}}$.
- In general, if **q** is a point, then the line $\vec{\mathbf{v}} \cdot \vec{\mathbf{x}} = \vec{\mathbf{v}} \cdot \vec{\mathbf{q}}$ is passing through q and orthogonal to $\vec{\mathbf{v}}$.
- In higher dimensions, all stay the analogous. $\vec{\mathbf{x}} = (x, y, z)$. Fix $\vec{\mathbf{v}} = (a, b, c)$. The set of points $\ell := \{\vec{\mathbf{x}} \in \mathbb{R}^3 \mid \vec{\mathbf{v}} \cdot \vec{\mathbf{x}} = \mathbf{0}\}$ form a plane in 3D.

In many cases, we can think about a vector as a point and vice versa.



The Diet Problem as an LP problem

- We will denote by x the number of bananas we consume per day.
- We will denote by y the number of bananas we consume per day.
- These x and y **are the only unknown, and what we need to optimize.**

$\vec{\mathbf{x}} = (x, y) = (\#\text{bananas/day}, \#\text{oranges/day})$

- For a diet to be healthy, we need to get a sufficient dose (quantity in grams) of each type vitamins. Assume n types of vitamins 1…n
- ***Given:*** $a_{i,1}$ – the amount of vitamin *i* in banana. $a_{i,2}$ the amount of vitamin *i* in an orange.

$\vec{\mathbf{a}}_i = (a_{i,1}, a_{i,2})$

- **Given:** b_i – minimum required daily dose of vitamin i *(i=1..n)*

$$\vec{\mathbf{a}}_i \cdot \vec{\mathbf{x}} = \underbrace{a_{i,1} \cdot x}_{\text{vitamin i from bananas}} + \underbrace{a_{i,2} \cdot y}_{\text{vitamin i from oranges}} \geq b_i$$

$$\begin{aligned} a_{11}x + a_{12}y &\geq b_1 \\ &\vdots \\ a_{n1}x + a_{n2}y &\geq b_n \end{aligned}$$

- ***Given:*** c_1 – the cost of a banana (dollars/unit). And given: c_2 the cost of an orange.
- $\vec{\mathbf{c}} = (c_1, c_2)$ *is the cost vector*
- *The daily cost of our diet is*

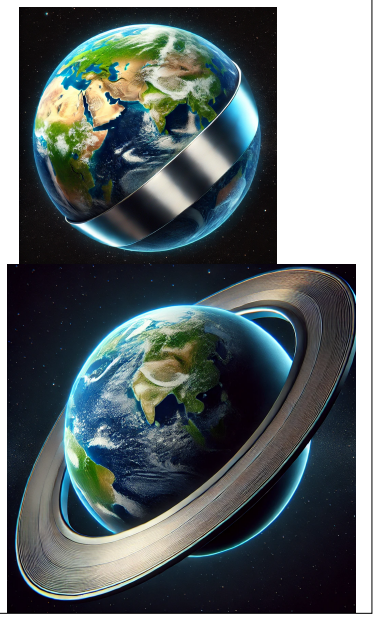
$$\vec{\mathbf{c}} \cdot \vec{\mathbf{x}} = \underbrace{c_1 \cdot x}_{\text{daily cost for bananas}} + \underbrace{a_{i,2} \cdot y}_{\text{cost for oranges}}$$

link

Minimize: minimize the cost of a healthy diet

The earth and the cat problem

- Question;
  - Place a rope around the equator.
  - Add 6ft to the rope
  - and arrange the rope concentrically.
- Could a cat pass in the gap between the equator and the rope ?
- Ps Earth Circumference 40,075 kilometers.



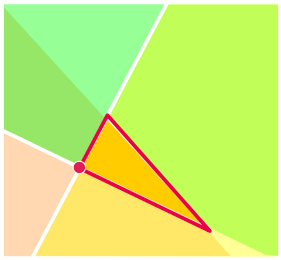
Linear Programming – The Geometry

- Each constraint defines defines a half-space region in *d*-dimensional space.
- The *feasible region* is the (convex) intersection of these half-spaces.

- We will treat the case $d = 2$, where each constraint defines a *half-plane*.
- The equation y=ax+b defines a line, which we could also write as (-a)x+(1)y=b. Pointed one one side of this line forms a half-plane.

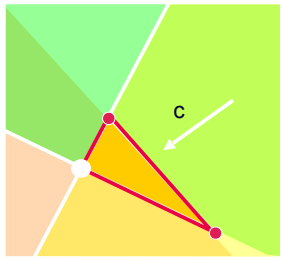
$$a_1x + a_2y \geq b$$

$$a_1x + a_2y \leq b$$



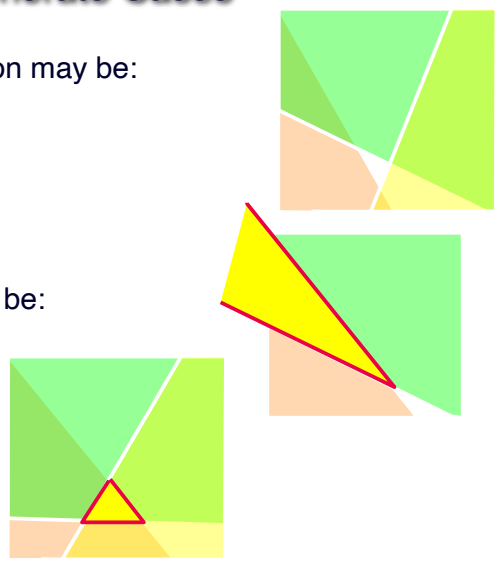
More Geometry

- The solution to the linear program is a point in the feasible region that is extreme in the direction of the target function.
- **Theorem:** Any bounded linear program that is feasible has **a** solution, which is a *vertex* of the feasible region.
- **Proof:** Convexity and cats
  - Pick a point $\vec{p}_1 = \arg\min\{\vec{c} \cdot \vec{x} \mid \vec{x} \in C\}$
  - If p_1 is not a vertex, then there is a direction $\vec{d}$ such that both $\vec{p}_2 = \vec{p}_1 + \varepsilon\vec{d}, \in C$, and $\vec{p}_2' = \vec{p}_1 - \varepsilon\vec{d} \in C$
  - (for a small enough $\varepsilon > 0$.
  - Assume WLOG $\vec{c} \cdot \vec{p}_2 = \vec{c} \cdot \vec{p}_1$. By controlling ε we can move $\vec{p}_2$ to be on a lower-degree feature than $\vec{p}_1$ (for example face=>edge, edge=>vertex etc.
  - Question: What if $\vec{c} \cdot \vec{p}_2 > \vec{c} \cdot \vec{p}_1$?



Degenerate Cases

- The feasible region may be:
  - Empty
  - Unbounded
- The solution may be:
  - Not unique



Solving the Linear Programming Problem

- The solution we mean - find $\arg\max\{\vec{c}\cdot\vec{x} \mid \vec{x}\in C\}$
- Several algorithms:
  - In $\mathbb{R}^2$ (only Bananas and Oranges) We could construct C explicitly by calculating all its vertices (lets do it in $O(n^3)$). In $\mathbb{R}^2$, there are $\leq n$ vertices.
  - Evaluate each vertex v_i to estimate $\vec{c}\cdot v_i$ and pick the best v_i .
  - In $\mathbb{R}^3$ (add avocado to the diet) Could use the same approach. Time is $O(n^4)$ (why?)
  - $\mathbb{R}^d, d\geq 4$ … Not a good idea to construct all vertices of C, since might have $\Omega(n^{\lfloor d/2\rfloor})$ vertices. Avoid constructing all vertices
  - $\mathbb{R}^d$, $2\leq d\leq 10$ use incremental randomized algorithm
  - $\mathbb{R}^d$, any d. Use the **Simplex** Algorithm. A very clever Steepest Decent Alg. Might be exponential (but very rarely)
- Common Variant's
  - Some vars needs to be integers (**ILP**)
  - The number of constrains is huge, but we could do with only a small subset (separating hyperplanes techniques, column generation)
  - Sometimes we care about the **Dual LP** - it gives some indications when near opt.
  - Sometimes the cost function and/or constrains Convex Programming - slightly harder than LP, much easier than non-convex optimization.
  - Steepest Decent, Lagrange multipliers - as time permit

Linear Programming in d dimension - Example

- Define: (amount amount consumed per day)
  - j – types of foods ($1\leq i\leq$**d**). *($i=1\rightarrow$ banana, $i=2\rightarrow$ oranges, $i=3\rightarrow$ avocado…) This is the dimension of the LP problem.*
  - x_j - the amount of food j consumed daily $1\leq j\leq d$)
    (these are the d unknowns that we need to optimize)
    $\vec{\mathbf{x}}=(x_1,x_2\ldots x_d)$
  - i – types of vitamins ($1\leq i\leq n$).
  - a_{ij} – the amount of vitamin j in one unit of food i.
    $\vec{\mathbf{a_i}}=(a_{i,1},a_{i,2}\ldots a_{i,d})$
    $\vec{a_i}\cdot\vec{x}\geq b_1$
  - b_j – minimal daily dose for vitamin i. $(1\leq i\leq n)$
    - Constrains : $\forall i: a_{i,1}x_1+a_{i,2}x_2+\ldots a_{i,d}x_d\geq b_i$ for every i
    - same using vector notation: $\vec{\mathbf{a_i}}\cdot\vec{\mathbf{x}}\geq b_i$
  - c_i – the cost of a unit of food j $(1\leq j\leq d)$
    $\vec{\mathbf{c}}=(c_1,\ldots c_d)$
- **LP problem**

  minimize the cost $\vec{\mathbf{c}}\vec{\mathbf{x}}=\sum c_jx_j$

  Such that (s.t.)
  for every $1\leq i\leq n$
  $\vec{\mathbf{a_i}}\cdot\vec{\mathbf{x}}\geq b_i$

Minimize : c^Tx
Subject to : $Ax\geq b$

linprog

Solve linear programming problems

Syntax

```
x = linprog(f,A,b)
x = linprog(f,A,b,Aeq,beq)
x = linprog(f,A,b,Aeq,beq,lb,ub)
x = linprog(f,A,b,Aeq,beq,lb,ub,options)
x = linprog(problem)
[x,fval] = linprog(__)
[x,fval,exitflag,output] = linprog(__)
[x,fval,exitflag,output,lambda] = linprog(__)
```

Description

Linear programming solver

Finds the minimum of a problem specified by

$$\min_x f^Tx \text{ such that } \begin{cases} A\cdot x\leq b, \\ Aeq\cdot x=beq, \\ lb\leq x\leq ub. \end{cases}$$

f, x, b, beq, lb, and ub are vectors, and A and Aeq are matrices.

Do we really need to solve LP in higher dimension

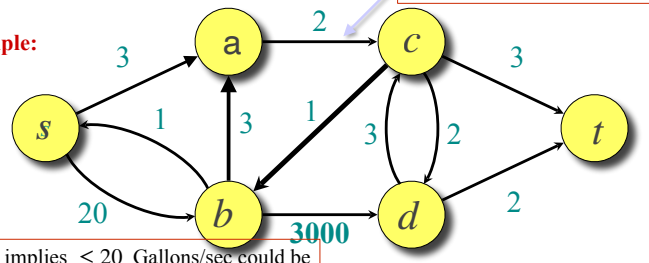
- Ok, diets are important, and it would be nice to include more ingredients than Oranges and Bananas, but are there other applications to high-dim LP ?

Max-Flow in Networks

- **Definition.** A ***flow network*** is a directed graph, with two distinguished vertices: a ***source*** s and a ***sink*** t. Each edge $(u, v) \in E$ is given with a nonnegative ***capacity*** $c(u, v)$.
- The values could specify the number of cars per minute on this road, or number of Gbyte on this link
- Goal: Send as many cars/bytes/gallons from s to t, without violating the edges capacities, and without violating the flow conservation (coming next)

$C(a, c) = 2$ implies that at most 2 cars/ gallon/GB could be sent along this link

Example:



$C(s, b) = 20$ implies ≤ 20 Gallons/sec could be sent directly $s \rightarrow b$

- Def: A solution to the flow network flow problem (or in short, the **flow**) on G is a set of values (numbers) $p(u, v) \geq 0$ specific for every edge $(u, v) \in E$.
- So for the example below, we need to specify the numbers $\{p(s, d), p(s, b), p(d, c), p(g, b) \ldots\}$ These are the unknown that we need to compute.
- To be a **legal** flow, these values must satisfy two sets of conditions:
  1. **Capacity constraint**: For all $(u, v) \in E$, $0 \leq p(u, v) \leq c(u, v)$ // don't overflow/bust any pipe
  2. **Flow conservation**: For all u ∈ V, which is not the source nor the sink

$$\sum_{(u, v_i) \in E} p(u, v_i) = \sum_{(v_i, u) \in E} p(v_i, u)$$

What comes in must go out. That is, every node is a memory-less router. It receives flow, and steers it to destinations.

The ***total value*** of a flow is the sum of the flow flows out of the source:

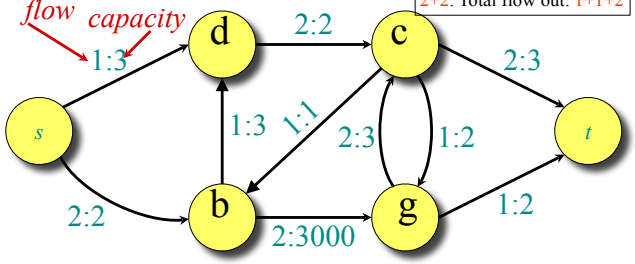
$$\sum_{(s,v) \in E} p(s, v)$$

In the example, the value of the flow equals 1+2=3.

Goal: Maximize total flow

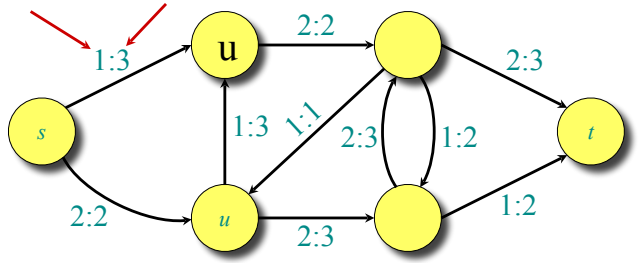
Note the format of two numbers next to each edge.

total flow into c 2+2. Total flow out: 1+1+2



A flow on a network

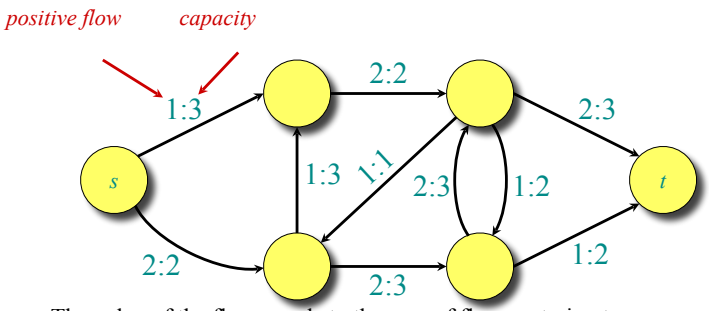
positive flow *capacity*



Flow conservation
- Flow into u is $2 + 1 = 3$.
- Flow out of u is $0 + 1 + 2 = 3$.

The value of this flow is $1 - 0 + 2 = 3$.

Lemma

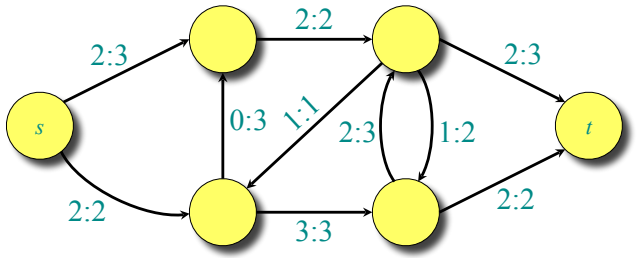


Lemma: The value of the flow equals to the sum of flows entering t

$$\sum_{v \in V} p(s,v) = \sum_{u \in V} p(v,t)$$

The maximum-flow problem

Maximum-flow problem: Given a flow network G, find a flow of maximum value on G.



The value of the maximum flow is 4.

- Linear programming and Network Flow problem
- We can use LP to find the max (legal) flow in the network
  - Input: Directed Graph $G(V,E)$, $s,t \in V$, and capacities on all edges (constants)
  - Variables: $f(u,v)$, $\forall (u,v) \in E$
  - Const Function $\max \sum_{(s,v)\in E} p(s,v)$
  - Constrains:

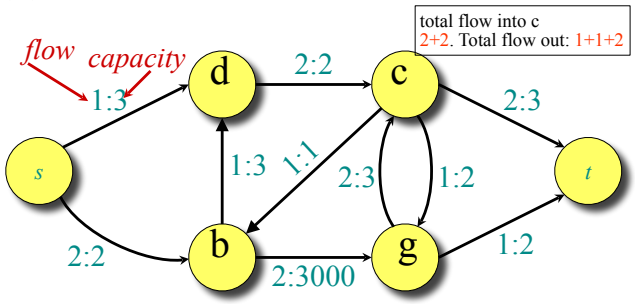
$$0 \le p(u,v) \le c(u,v), \quad \forall (u,v) \in E$$

$$\sum_{(u,v)\in E} p(u,v) - \sum_{(v,u)\in E} p(v,u) = 0. \quad \forall u \in V \setminus \{s,t\}$$

What comes in must go out. That is, every node is a memory-less router. It receives flow, and steers it to destinations.

The ***total value*** of a flow is the sum of the flow flows out of the source:
$\sum_{(s,v)\in E} p(s,v)$
In the example, the value of the flow equals 1+2=3.

Goal: Maximize total flow



What if we want integer solution ?

One solution - Ford-Fulkerson
Runs in O(m |F|), where |F| is the max value of the flow

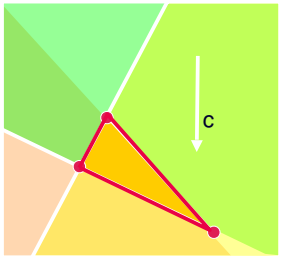
Another is to use ILP (next)

And many more (we will get back to it later in the course)

The Simplex Algorithm

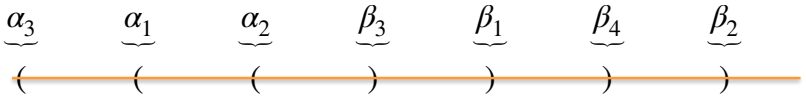
- Assume WLOG that the cost function points "downwards".
- Construct (some of) the vertices of the feasible region.
- Walk edge by edge downwards until reaching a local minimum (which is also a global minimum).

- In R^d, the number of vertices might be $\Theta(n^{\lfloor d/2 \rfloor})$.



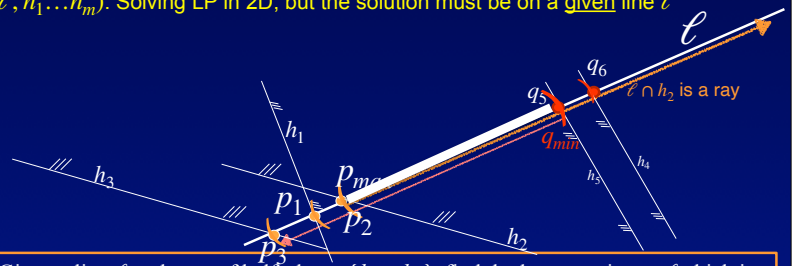
Toward a faster algorithm in small dimensions

- 1-dimensional linear programming
  - Given 2n constants (constrains) $\alpha_1, \alpha_2 \ldots \alpha_n, \quad \beta_1, \beta_2 \ldots \beta_n$ (not necessarily sorted)
  - find in O(n) time the minimum x such that
- $x \geq \alpha_i$ (for every $1 \leq i \leq n$) and $x \leq \beta_i$ (for every $1 \leq i \leq n$)
- What is the feasible region ? Could it be that the problem has no solution ?
- Answer



Feasible solution $\{x \mid \quad \max(\alpha_i) \leq x \leq \min \beta_j\}$

$1D - LP(\ell, h_1 \ldots h_m)$. Solving LP in 2D, but the solution must be on a given line ℓ



Problem: Given a line ℓ and a set of half-planes $\{h_1 \ldots h_n\}$, find the lowest point on ℓ which is inside $\cap_{i=1}^{m} h_i = h_1 \cap h_2 \cap \ldots \cap h_m$

- Each half-plane either contains the point $(0, +\infty)$ or contains the point $(0, -\infty)$.
- Consider first only half-plane containing $(0, +\infty)$.
- Let ℓ_i be the line bounding h_i. Compute $p_i = \ell \cap \ell_i$
- Let p_{max} be the highest such point (p_2 in the example). Any solution to the LP must be on the portion of ℓ above p_{max}
- basically shoot a ray from p_{max} upward along ℓ - if there is a solution, it is along this ray.

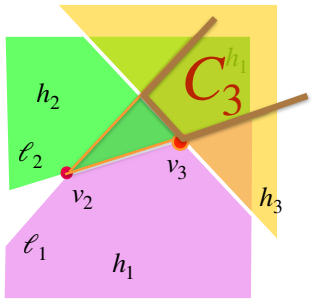
- Similarly, find the half-planes contain $(0, -\infty)$. Compute their intersections with ℓ. Let q_{min} be the lowest intersection points.
- Any solution to the LP which is on ℓ must be between p_{max} and q_{min}.
- Note that it is possible that qmin is below p_{max}. In this case, we have no solution on ℓ

O(n^2) Incremental Algorithm

- The idea:
  - Start by intersecting two halfplanes.
  - Add halfplanes one by one and update optimal vertex by solving one-dimensional LP problem on new line *if needed*.

Incremental Algorithm - Notation

- h_i is the i'th constrained half-plane
- ℓ_i is the line bounding h_i
- $C_i = h_1 \cap h_2 \cap \ldots h_i$ is the feasible region of the first i' constrains
- v_i is the optimal solution to the first i constrains - it is the lowest point of C_i



Cost function to minimize: c(x,y)=y .
Returns the lowermost point in feasible region

Incremental Algorithm Basic Theorem

- Theorem:
  1. if $v_{i-1} \in h_i$, then $v_i = v_{i-1}$. // O(1) check, nothing to do
  2. if $v_{i-1} \notin h_i$, then it is sufficient to look for v_i on ℓ_i *using 1DLP* (rather than searching in the whole plane)
- Conclusion: If there is no solution on ℓ_i , then there is no solution at all. The feasible region is empty.
- Proof:
  1. Trivial. Otherwise v_i would not have been optimum before.
  2. - in the next slide

Basic Theorem - case 2.

Recall v_i is the lowest point at $C_i = h_1 \cap h_2 \cap \ldots \cap h_i$

Assume that v_i is not on ℓ_i

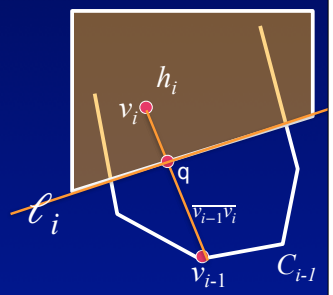
v_i must be in C_{i-1} By convexity, also the segment $\overline{v_{i-1}v_i}$ (from v_i to v_{i-1}) is in C_{i-1} .

Assume WLOG: Our cost function pushes us downward.

Consider point q: the intersection of the segment $\overline{v_{i-1}v_i}$ with l_i.

Notice: q is also in h_i, and in is C_{i-1}. It is lower than v_i

Contradicting the assumption that v_i is not on ℓ_i





Same theorem – in an algorithmic terms

Compute $C_i = h_1 \cap h_2$, and v_2

For i=3…n

{

1. Check if $v_{i-1} \in h_i$. If yes, then $v_i = v_{i-1}$. // O(1),

` ELSE

2. // v_i must be on the line ℓ_i call 1D-LP(ℓ_i, $h_1 \ldots h_{i-1}$)
3. If 1D-LP does not have a solution on ℓ_i - stop. There is no solution anywhere.

   set v_i to be the solution that 1D-LP found.

}

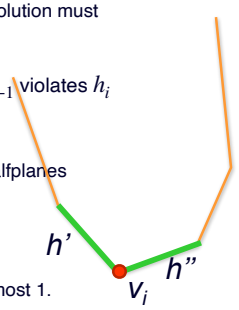
Complexity Analysis

- Worst case, each new constrain h_h forces solving a new 1DLP
- $T(n) = \sum_{i=3}^{n} c \cdot i = \Theta(n^2)$

Theorem : The expected time for the randomize version is O(n)

Backward analysis

- Recall that if v_{i-1} violates h_i then $v_i \in \ell_i$. In words, the new optimum solution must on the line bounding h_i.
- **Question**: What is the **probability** that at the i'th step of the algorithm, v_{i-1} violates h_i ? (that is $v_{i-1} \neq v_i$).
- **Answer**: Exactly $2/\mathbf{i}$. Here is the reason:
- v_i is determined by two half-planes. It does not care it which order the halfplanes were inserted.
- The probability that one of them is h_i is 2/i.
- Let y_i be a random var, which is 1 iff $v_{i-1} \neq v_i$
- The probability that h_i is one of the other halfplanes is $\frac{i-2}{i}$ which is almost 1.
- Let y_i be a random var, which is 1 iff $v_{i-1} \neq v_i$
- Total work $\sum_{i=3}^{n} 1 + i \cdot y_i$.
- Expected work $E\left(\sum_{i=3}^{n} 1 + i \cdot y_i\right) = n + \sum_{i=3}^{n} i \cdot Pr(y_i = 1) = n + i \cdot \frac{2}{i} = 3n$
- YAY.



Just to Make Sure …

- False Claim:
  - The probabilistic analysis is for the average input. Hence there exist bad sets of constraints for which the algorithm's expected runtime is *more* than O(*n*), and there exist good sets of constraints for which the algorithm's expected runtime is *less* than O(*n*).
- True Claim:
  - The probabilistic analysis is valid for *all* inputs. The expected complexity is over all *permutations* of this input.

LP in 3D

- Now the input is a collection of **half-spaces** $\{h_1 \ldots h_n\}$.

Now l_i is the plane bounding h_i. (notations are analogous to the 2D case).

We will define v_3 as the intersection of the **planes** l_1, l_2 and l_3.

We insert the other halfspaces $\{h_4 \ldots h_n\}$ at a random order, and update v_i according to the following Theorem:

- Theorem:

1. if $v_{i-1} \in h_i$, then $v_i = v_{i-1}$. // O(1) check,
   nothing to do
2. if $v_{i-1} \notin h_i$, then the solution (if exists) is on l_i .
   run v_i = 2DLP($h_1 \cap l_i, h_2 \cap l_i$, $h_3 \cap l_i, \ldots, h_{i-1} \cap l_i$).
   Terminates if there is no solution (that is, $C_i = \varnothing$)

LP in 3D and higher dimension

In 3D, the worst case running time is $\Theta(n^3)$ ***(prove).***

However, the expected running time is O(n). In general, the running time in d-dimension is O(d! n). That is, linear in any fixed (and small) dimension.

Integer Linear Programming (ILP)

- Linear programming problems at which values of the computed variables must be integers are called *Integer Linear Programming (ILP)* problems.
- If only some of the variables have to integers, we call them ***Mixed*** *Integer Linear Programming* problems.
- There is a huge number of problems that could be phrased as ILP.

  (include many NP-hard problems, where no polynomial-time algorithms exist)
- A few libraries could handle them, including CPLEX.
- Running time could varies a lot, and could be extremely slow for some instances.
- Yet extremely useful for instances when actual running time is acceptable.
- Also useful for comparing fast heurists to global optimum.

Integer Linear Programming (ILP)
Example in Two Dimensions

- Define: (amount consumed per day)
  - types of foods : {oranges, bananas}
  - j – types of vitamins ($1 \le j \le n$).
  - x – number of pounds of oranges we recommend daily
  - y – number of pounds of bananas we recommend daily
    // these are the only unknown we have to compute.
  - a_{ji} – the amount of vitamin j in a unit of food i
    - *(i=1* for oranges, *i=2* for bananas)
  - c_1 – the number of calories in an orange.
  - c_2 – the number of calories in a banana.
  - b_j – minimal daily required amount of vitamin j.
- Constraints (we need to consume some minimal amount of each vitamin):

Another constrain: both *x,y* in the solution are integers.

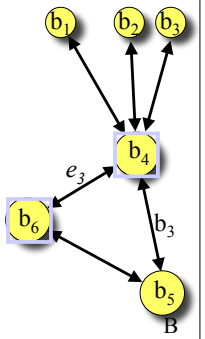
Now we have ILP problem.

Minimize: the total number of calories consumed:

$$C((x,y)) = c_1 x + c_2 y$$

$$a_{11}x + a_{12}y \ge b_1$$
$$\vdots$$
$$a_{n1}x + a_{n2}y \ge b_n$$

Vertex Cover and ILP



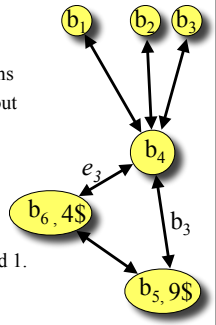
- Given: A graph G(V,E). A subset $C \subseteq V$ is a *vertex cover* if every edge$(u, v) \in E$ we have either $u \in C$ or $v \in C$ or both
- Finding the **min-cardinality** Vertex Cover is NP-Hard
- ILP for this problem: the variables are $x_1 \ldots x_n$. All are integers and between 0 and 1.
- $v_i \in C$ iff $x_i = 1$ (for $i = 1 \ldots n$)

s.t.

$$x_i + x_j \ge 1 \qquad \forall (v_i, v_j) \in E$$

$$minimize \sum_{i=1}^{n} x_i$$

Min-Weight Vertex Cover and ILP



- Sometimes the LP (instead of the ILP) could help us finding good approximations
- Given: A graph G(V,E). Each vertex v_i is given with a weight $w_i > 0$. Think about it as the cost of this vertex.
- A subset $C \subseteq V$ is a *vertex cover* if every edge$(u, v) \in E$ we have either $u \in C$ or $v \in C$ or both
- The cost of C is the sum of weights of vertices in C.
- Finding the **min-cardinality** Vertex Cover is NP-Hard
- ILP for this problem: the variables are $x_1 \ldots x_n$. All are integers and between 0 and 1.
- $v_i \in C$ iff $x_i = 1$ (for $i = 1 \ldots n$)

$$minimize \sum_{i=1}^{n} w_i x_i$$

s.t.

- $0 \le x_i \le 1$ and an integet, for every x_i
- $x_i + x_j \ge 1 \quad \forall (v_i, v_j) \in E$

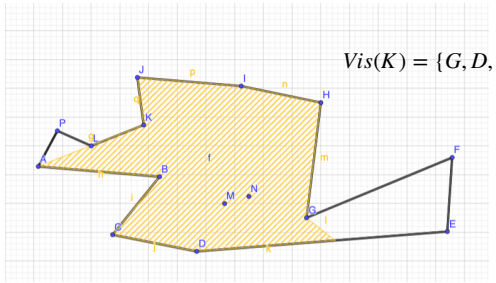
Art Gallery - on the board

- Given a polygon, find a subset of the vertices that sees every other vertex
- Let ***Vis(i)*** be the set of vertices that vertex i sees. $Vis(K) = \{G, D, C, A, K, J, I, H\}$
- For a vertex v_i we set x_i=1 if we place a guard at v_i. Otherwise $v_i = 0$
- As usual , x_i are integers between 0 to 1.

$$minimize \sum_{i=1}^{n} x_i$$

s.t.

$$\sum_{k \in Vis(i)} x_k \ge 1 \qquad \forall 1 \le i \le n$$



$Vis(K) = \{G, D, C, A, K, J, I, H\}$